

Entonces las dir. lineales 0 a 4MB^(excluyendo) se corresponden siempre a espacio "kernel" mapeado para todas las tareas (área compartida) y 4MB a 4MB+24KB se corresponde a las direcciones TJ a TJ+24KB físicas.

IDT:

#	seg. sel	offset	DPL
1-31 (reservados -excepciones)	$0 \times 1 \ll 3$	RET-AT-EX	0
32 (clock)	$0 \times 1 \ll 3$	RET-AT-CLOCK	0
47 (control Tarea)	$0 \times 1 \ll 3$	RET-AT-47	3
48 (dameRet)	$0 \times 1 \ll 3$	RET-AT-48	3

donde las etiquetas \uparrow son los offsets dentro de área que nombré "kernel" donde se encuentran estas rutinas de atención. Suponiendo que le dedico una página (podría ser menos) a cada una, sus valores estarán entre 0 y 4MB-4KB y no se superpondrán.

4/6

TSS de la

TSS: los campos de la $\sqrt{\text{tarea } j}$ (donde $1 \leq j \leq m$) tendrán el siguiente aspecto si la tarea es de tipo A.

CS : $0 \times 2 \ll 3$

GS, ES, FS, DS, SS : $0 \times 4 \ll 3$

Registros de propósito general (menos ESP, EBP) : 0

ESP y EBP : $4\text{MB} + 12\text{KB}$

EFLAGS : 0×202 (int. activadas)

EIP : 4MB

CR3 : el offset dentro de kernel que corresponde al PD de la tarea JA
(OFF_PDJA)

SS1 : $0 \times 4 \ll 3$

ESP1 : $4\text{MB} + 12\text{KB}$

SS0 : $0 \times 3 \ll 3$

ESPO : $4\text{MB} + 8\text{KB}$

(SS2 y ESP2 no corresponden porque hay solo 2 niveles de protección)

Si la tarea fuera de TIPO B, habría que sumarle 12KB a ESP, EBP, EIP, ESP1 y ESPO, y CR3 sería OFF_PDJB (por más que sus esquemas de paginación sean idénticos, le asigne a cada una sus propios PD y PT)



offset: dd 0
selector: dw 0

por más que asumí que siempre se ejecuta alguna de las dos tareas del por lo dejó así.

b) -isr32:

pushad

call pic-finish1

call sched-Next

str cx

cmp ax, cx

je -fin

mov [selector], ax

jmp far [offset]

-fin:

popad

iret

uint8_t sched[n];

+ss_t +ss[2n];

uint16_t ultima; //entre 0 y n-1

uint16_t sched-Next() {

uint16_t res = 0;

ultima = (ultima + 1) % n

bool found = false;

for (i = 0; i < n && !found; i++) {

uint8_t estado = sched[(ultima + i) % n];

if (estado != 0) {

found = true

res = (ultima + i) % n + 5;

if (estado == 2)

res += n;

}

} return res < 3;



rut-excepción: (para alguna excepción)

pushad

push esp

call excep-handler

add esp, 4

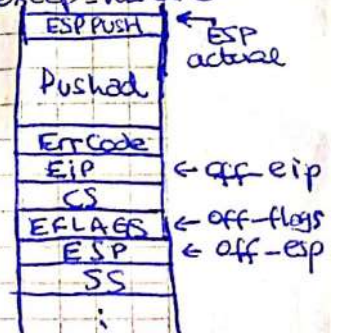
popad

iret

observación: hay cambio de privilegio cuando ocurre la interrupción.



Stack al llamar a excep-handler



//restaura solo los valores necesarios!
void excep_handler(uint32_t esp_actual) {

uint8_t tipo = sched[ultima] - 1;

+ss_t tarea +ss = +ss[ultima + n * tipo];

restaurarMemoriaDeLaTarea(&(tarea +ss - cr3));

esp_actual[off-eip] = 4MB + 12KB * tipo;

esp_actual[off-esp] = 4MB + 12KB + 12KB * tipo;

esp_actual[off-ebp] = 4MB + 12KB + 12KB * tipo;

esp_actual[off-flags] = 0x202;

}

//OFF-FBP está dentro de pushad



5/6

c) -isr47: (cambiarTarea)

call cambiarTarea

INT 32

iret

// llamo a int. de reloj para que pare a ejecutarse otra tarea (detiene la actual)

void cambiarTarea() {

if (sched[ultima] == 1)

 sched[ultima] = 2;

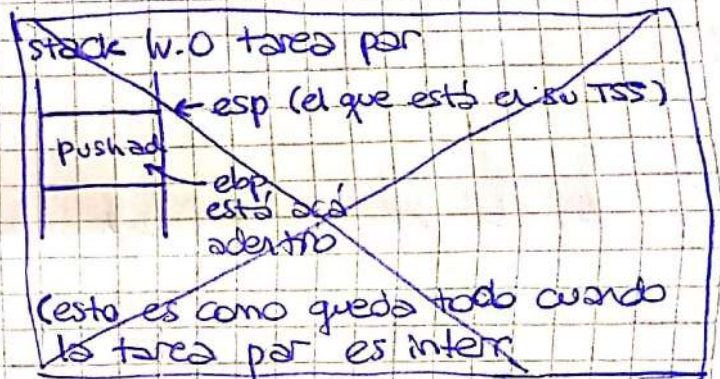
else

 sched[ultima] = 1;

}

-isr48: (dameRet)

; la tarea par deja de ser
; ejecutada cuando se llama
; a cambiarTarea; en particu-
; lar cuando la interrupción de
; reloj que invoca ésta realiza el jmp far.
; el EBP no es modificado en ningún momento, por lo que
; puedo buscarlo en la TSS sin problema.



(si la tarea nunca fue ejecutada el ebp igual está en su valor inicial así que esto vale)

call buscar dameRet

iret

ojo: En este caso también es necesario preservar el valor de los registros con pushad/popad, por lo que para devolver el resultado hay que multiplicar la posición en la pila en que se apila EAX

uint32_t dameRet() {

uint8_t tipo = sched[ultima] - 1; // tarea A = 0
// tarea B = 1

+tss - tareaPar - tss = tss[ultima + n * (!tipo)];

uint32_t* ebp_p = (uint32_t*) tareaPar - tss - ebp;

return *(ebp_p + 1); → Nota: El contenido de [EBP+1] puede variar según el mapa de paginación de la tarea dada por lo que es necesario antes de hacer la desreferenciación, cambiar el cr3 por el de la tarea.

}

6/6

③ ~~obs: una entrada de IDT mide 8B.~~

- En IDTR se guardan la dirección base de la IDT y el límite (tamaño de la table, es decir primer offset no direccional de esta, $n * \text{size(idte)}$ si n es la cantidad de descriptores de interrupción).

↳ Nota: El límite es el último byte direccionable de la tabla

```
uint32_t countUserInts(idt_descriptor* idt_desc) {
    idt_entry* actual = (idt_entry*) idt_desc->idt_base;
    idt_entry* finIdt = (idt_entry*) (idt_desc->idt_base +
                                     idt_desc->idt_limit);
```

```
uint32_t res = 0;
```

```
while (actual != finIdt) {
```

```
    if (actual->dpl >= 2)
```

```
        res++;
```

```
        actual++; // como actual es un puntero a la estructura,
                  // esta suma me lleva directo a la siguiente entrada.
```

```
    return res;
```

```
}
```

- El límite del descriptor de gdt funciona igual que el de idtr.

✓

// funciona de manera similar a ②

b) `uint32_t countLevelZeroTasks (gdt_descriptor* gdtDesc){`

`gdt_entry* base = (gdt_entry*) gdtDesc->gdt_base;`

`gdt_entry* finGDT = (gdt_entry*)(gdtDesc->gdt_base +`
`gdtDesc->gdt_limit)`

`uint32_t res = 0;`

`while (actual != finGDT){`

Nota: las entradas de GDT no poseen este campo

`uint8_t DPLCS es 0 = DPLEscero ((actual->cs) >> 3, base)`

`if (DPLCS es 0)`

`res++;`

`actual++;`

`}`

`return res;`

`uint8_t DPLEs 0 (uint32_t index-GDT, gdt_entry* gdt){`

`gdt_entry* entry = gdt + index-GDT;`

`uint8_t res = 0;`

`if (entry->dpl == 0)`

`res = 1;`

`return res;`

✓